

Online Power-Managing Strategy with Hard Real-Time Guarantees

Jian-Jia Chen^{*}, Mong-Jen Kao⁺, D.T. Lee⁺, Ignaz Rutter^{*}, and Dorothea Wagner^{*}

^{*}Faculty for Informatics, Karlsruhe Institute of Technology (KIT), Germany.

Email: j.chen@kit.edu, rutter@kit.edu, dorothea.wagner@kit.edu

⁺Research Center for Infor. Tech. Innovation, Academia Sinica, Taiwan.

Email: mong@citi.sinica.edu.tw, dtlee@ieee.org

April 9, 2013

Abstract

We consider the problem of online dynamic power management that provides hard real-time guarantees. In this problem, each of the given jobs is associated with an arrival time, a deadline, and an execution time, and the objective is to decide a schedule of the jobs as well as a sequence of state transitions on the processors so as to minimize the total energy consumption. In this paper, we examine the problem complexity and provide online strategies to achieve energy-efficiency. First, we show that the competitive factor of any online algorithm for this problem is at least 2.06. Then we present an online algorithm which gives a 4-competitive schedule. When the execution times of the jobs are unit, we show that the competitive factor improves to 3.59. At the end, the algorithm is generalized to allow a trade-off between the number of processors we use and the energy-efficiency of the resulting schedule.

1 Introduction

Reducing power consumption and improving energy efficiency has become an important designing requirement in computing systems. For mobile devices, efficient energy management can effectively extend the standby period and prolong battery lifetime. For large-scale computing clusters, appropriate power-down mechanism for idling processing units can considerably reduce the electricity bill.

In order to increase the energy efficiency, two different mechanisms have been introduced to reduce the energy consumed for idling periods. (1) *Power-down Mechanism*: When a processor is idling, it can be put into a low-power state, e.g., sleep or power-off. While the processor consumes less energy in these states, a fixed amount of energy is required to switch the system back to work. In the literature, the problem of deciding the sequence of state transitions is referred to as *dynamic power management*. (2) *Dynamic Speed Scaling*: The concept of dynamic speed scaling refers to the flexibility provided by a processor to adjust its processing speed dynamically. The rate of energy consumption is typically described by a convex function of the processing speed. This feature is also referred to as *dynamic voltage frequency scaling*, following its practical implementation scheme.

For systems that support the power-down mechanism, Baptiste [3] proposed the first polynomial-time algorithm to decide the optimal strategy for turning on and powering off the system for aperiodic real-time jobs with unit execution time. In a follow-up paper, Baptiste et al. [4] further extended the result to jobs of arbitrary execution time and reduced the time complexity. When

strict real-time guarantees are not required, i.e., deadline misses of jobs are allowed, Augustine et al. [2] considered systems with multiple low-power states and provided online algorithms. A simplified version of this problem is also known as *ski-rental* [17].

Dynamic speed scaling was introduced to allow computing systems to reach a balance between high performance and low energy consumption dynamically. Hence, scheduling algorithms that assume dynamic speed scaling, e.g., Yao et al. [19], usually execute jobs as slowly as possible while ensuring that timing constraints are met. When the energy required to keep the processor active is not negligible, however, executing jobs too slow may result in more energy consumption. For most realistic power-consumption functions, there exist a *critical speed*, which is most energy-efficient for job execution [6, 14].

Irani et al. [14] initiated the study of combining both mechanisms. For offline energy-minimization, they presented a 2-approximation. For the online version, they introduced a *greedy procrastinating principle*, which enables any online algorithm for speed scaling without power-down mechanism to additionally support the power-down mechanism. The idea behind this principle is to postpone job execution as much as possible in order to bundle workload for batch execution. The usage of job procrastination with dynamic speed scaling for period tasks has later been explored extensively in a series of research [5, 6, 18].

The combination of the power-down mechanism with dynamic speed scaling suggests the philosophy of *racing-to-idle*: Execute jobs at higher speeds and gain longer quality sleeping intervals. In a recent result, Albers and Antoniadis [1] show that the problem of minimizing the energy consumption for speed scaling with a sleep state is NP-hard and provide a $\frac{4}{3}$ -approximation.

Our Focus and Contribution. In this paper, we examine the problem of online dynamic power management to minimize the energy consumption. We present both lower bounds on the problem complexity and algorithmic results.

First, we show that the competitive factor of any online algorithm for this problem is at least 2.06. This shows that this problem is already harder than the ski-rental problem, which has a tight competitive factor of 2 that is inherited by a couple of online scheduling problems as the only known lower bound [15, 17].

Second, we present a 4-competitive online algorithm that uses at most two processors for any given set of jobs known in advance to be packable on a processor. When the execution times of the jobs are unit, we show that the competitive factor improves to 3.59. Then, we generalize our algorithm for a prescribed collection of job streams to allow a trade-off between the number of processors we use and the energy-efficiency of the resulting schedule.

Note that, our assumption on the input job set is crucial in the sense that packing the jobs using a given number of processors is known to be a long-standing difficult problem even for the offline case [8–10, 12], and for the online version only very special cases were studied [10, 16].

Due to the space limit, some technical details and proofs are provided in the appendix for further reference.

2 Notations and Problem Definition

In this section, we provide definitions to the scheduling model assumed in this paper, followed by a formal problem definition.

The Jobs. Each job j is associated with three parameters, namely, the arrival time a_j , the execution time c_j , and the deadline d_j . The arrival time of a job is the moment it arrives to the system and becomes ready for execution. The execution time is the amount of time it requires to finish its task, and the deadline is the latest moment at which the task must be completed. We assume that c_j and d_j are both known at the moment when j arrives to the system.

For notational brevity, for any job j , we use a triple $j = (a_j, d_j, c_j)$ to denote the corresponding parameters of j . We call a job j a unit job if $c_j = 1$ and we write $j = (a_j, d_j)$. Moreover, a job j is said to be *urgent* if $c_j = d_j - a_j$.

Energy Consumption. In this paper, processors for executing jobs are assumed to have three states, namely *busy*, *standby*, and *off*. When a processor is off, it cannot execute jobs and consumes a negligible amount of energy. Switching a processor from off to other states requires E_w units of energy. A processor is in busy state when executing a job. The amount of energy it consumes per unit of time when busy is denoted by ψ_b . When a processor is in standby, it consumes ψ_σ amount of energy per unit of time. We assume $\psi_\sigma \leq \psi_b$. For convenience, we use the terminology “turning on” and “turning off” to denote the transition between the off state and other states.

Provided the above notion, the *break-even time*, denoted by \mathcal{B} , is defined as E_w/ψ_σ . Literally, this corresponds to the amount of time a processor has to stay in standby in order to have the same energy consumption as a turn-on operation. Break-even time is an important concept that has been widely used for ski-rental-related problems [17] and dynamic power management algorithms in the literature, e.g., [13–15].

Job Scheduling. Let \mathcal{J} be a set of jobs. A schedule \mathbf{S} for \mathcal{J} on a set of processors \mathcal{M} is to decide for each processor $m \in \mathcal{M}$: (1) a set \mathcal{I}_m of time intervals during which processor m is turned on, and (2) a function $\text{job}_m(t): \mathbb{R}^+ \rightarrow \mathcal{J}$ of time indicating the job to occupy processor m at time t . The schedule \mathbf{S} is said to be *feasible* if for each job $j \in \mathcal{J}$, there exist a processor $m \in \mathcal{M}$ such that

$$\sum_{I \in \mathcal{I}_m} \int_{I \cap [a_j, d_j]} \delta(\text{job}_m(t), j) \cdot dt \geq c_j,$$

where $\delta(x, y) = 1$ if $x = y$ and $\delta(x, y) = 0$ otherwise. The energy consumption of the schedule \mathbf{S} , denoted $E(\mathbf{S})$, is hence

$$E(\mathbf{S}) = \sum_{m \in \mathcal{M}} \left(E_w \cdot |\mathcal{I}_m| + \int_{I \in \mathcal{I}_m} (E_b - \delta(\text{job}_m(t), \phi) \cdot (E_b - E_s)) \cdot dt \right).$$

The goal of the *Power-Minimizing Scheduling Problem* is to find a feasible schedule \mathbf{S} such that $E(\mathbf{S})$ is minimized.

In this paper, we consider the case where the jobs are arriving to the system dynamically in an online setting, i.e., at any time t , we can only see the jobs whose arrival times are less than or equal to t , and the scheduling decisions have to be made without prior knowledge on future job arrivals. To be more precise, let \mathcal{J} be the input job set and $\mathcal{J}(t) = \{j: j \in \mathcal{J}, a_j \leq t\}$ be the subset of \mathcal{J} that consists of the jobs whose arrival times are no greater than t .

Definition 1 (Online Power-Minimizing Scheduling). For any given set \mathcal{J} of jobs, the *online power-minimizing scheduling* problem is to compute a feasible schedule such that the energy consumed up to time t is small with respect to $OPT(\mathcal{J}(t))$, where $OPT(\mathcal{J}(t))$ is the energy consumed by an optimal schedule of $\mathcal{J}(t)$, for any $t \geq 0$.

The Schedulability of the Jobs. Chetto et al. [7] studied the schedulability of any given set of jobs and proved the following lemma.

Lemma 1 (Chetto et al. [7]). For any set \mathcal{J} of jobs, \mathcal{J} can be scheduled on one processor using the *earliest-deadline-first (EDF)* principle, which always selects the job with earliest deadline for execution at any moment, if and only if the following condition holds:

$$\text{For any time interval } (\ell, r), \text{ we have } \sum_{j: j \in \mathcal{J}, \ell \leq a_j, d_j \leq r} c_j \leq r - \ell. \quad (1)$$

Note that, it is well-known that, for any set \mathcal{J} of jobs, if there exists a feasible schedule for \mathcal{J} that uses only one processor, then the EDF principle is guaranteed to produce a feasible schedule [11]. Therefore, Condition (1) gives a necessary and sufficient condition for any set of jobs to be able to be packable on a processor.

3 Problem Lower Bound

In this section, we prove a lower bound of 2.06 on the competitive factor of any online algorithm by designing an online adversary \mathcal{A} that observes the behavior of the scheduling algorithm to determine the forthcoming job sequence.

Let Π be an online scheduling algorithm for this problem. We set $\psi_b = \psi_\sigma = \psi = 1$ and $E_w = k$, where k is an integer chosen to be sufficiently large. Hence the break-even time \mathcal{B} is also k . Without loss of generality, we assume a length of the minimum tick of the system to be ϵ_0 , which we further assume to be 1. We define a *monitor* operation of the adversary \mathcal{A} as follows.

Definition 2. When \mathcal{A} **monitors** Π during time interval $[t_0, t_1]$, it checks if Π keeps at least one processor on between time t_0 and t_1 . If \mathcal{S} turns off all the processors at some point t between t_0 and t_1 , then \mathcal{A} releases an urgent job of length ϵ_0 immediately at time $t + \epsilon_0$, forcing Π to turn on at least one processor to process it. If Π keeps at least one processor on during the monitored period, then \mathcal{A} does nothing.

Let x , η , and χ , where $0 \leq x \leq \frac{2}{5}$, be three non-negative parameters to be chosen carefully. The online adversary works as follows. At the beginning, say, at time 0, \mathcal{A} releases a unit job $(0, \mathcal{B}, \epsilon_0)$ and observes the behavior of Π . Let t be the moment at which Π schedules this job to execute. Since Π produces a feasible schedule, we know that $0 \leq t \leq \mathcal{B} - 1$. We have the following two cases.

Case(1): If $0 \leq t \leq (\frac{1}{2} - x)\mathcal{B}$, then \mathcal{A} monitors Π from time t to $\frac{3}{2}\mathcal{B}$.

Case(2): If j is not executed till $(\frac{1}{2} - x)\mathcal{B}$, \mathcal{A} releases $(\frac{1}{2} + x)\mathcal{B} - \epsilon_0$ unit jobs with absolute deadline \mathcal{B} at time $(\frac{1}{2} - x)\mathcal{B} + \epsilon_0$. As a result, the online algorithm is forced to wake up at least two processors to meet the deadlines of the jobs. The adversary continues to monitor Π until time $(\frac{3}{2} + \eta)\mathcal{B}$. If no urgent unit jobs have been released till time $(\frac{3}{2} + \eta)\mathcal{B}$, \mathcal{A} terminates. Otherwise, it monitors Π for another $\chi\mathcal{B}$ units of time until $(\frac{3}{2} + \eta + \chi)\mathcal{B}$.

Let $\mathcal{E}(\Pi)$ and $\mathcal{E}(\mathcal{O})$ denote the energy consumed by algorithm Π and an offline optimal schedule on the input sequence generated by \mathcal{A} , respectively. The following lemmas give a lower bound on the ratio $\mathcal{E}(\Pi)/\mathcal{E}(\mathcal{O})$ for each of the aforementioned cases, expressed as a function of k , x , η , and χ .

Lemma 2. If the first job released by \mathcal{A} is scheduled at time t with $0 \leq t \leq (\frac{1}{2} - x)\mathcal{B}$, then we have

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq 2 + \frac{1}{2}x - O\left(\frac{1}{k}\right), \quad \text{for any } 0 \leq x \leq \frac{2}{5}.$$

Lemma 3. If the first job released by the adversary is not executed until time $(\frac{1}{2} - x)\mathcal{B}$, then for any $x, \eta, \chi \geq 0$ we have

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \min \left\{ \frac{3 + x + \eta}{\frac{3}{2} + x}, \frac{4 + x + \eta + \chi}{2 + x + \eta}, \frac{5 + x + \eta + \chi}{2 + x + \eta + \chi} \right\} - O\left(\frac{1}{k}\right).$$

By combining Lemma 2 and Lemma 3 we obtain the following theorem.

Theorem 4. The competitive factor of any online algorithm for the online power-minimizing scheduling problem is at least 2.06.

4 Online Scheduling

We have seen in the last section that, in order to bundle the execution of the jobs while providing strict real-time guarantees, additional processors are necessary compared to those required by an optimal offline schedule.

In this section, we first assume that Condition (1) from Lemma 1 holds for the input set of jobs and present an online strategy that gives an energy-efficient schedule using at most two processors. In § 4.4, we generalize our algorithm for a prescribed collection of job streams, each of which delivers a set of jobs satisfying Condition (1), to allow a trade-off between the number of processors we use and the energy-efficiency of the resulting schedule.

We begin our discussion with a review on the commonly used approaches and their drawbacks in our problem model.

Common Approaches and Bad Examples. A commonly used approach to bundling the workload is to delay the execution of the jobs as long as possible until no more space for delaying the job execution is left, followed by using the earliest-deadline-first principle to schedule the jobs, e.g., the algorithm due to Irani et al. [14]. When there is no more job to execute, the ski-rental problem and related scheduling problems [17] suggest that we stay in standby for \mathcal{B} amount of time before turning off the processor. Let \mathcal{L} denote this approach.

Lemma 5. The competitive factor of \mathcal{L} can be arbitrarily large. Furthermore, even when $c_j = 1$ for all $j \in \mathcal{J}$, the competitive factor of \mathcal{L} is still at least 6.

The above lemma shows that, when the jobs can have arbitrary execution times, packing of the jobs should be done more carefully. Furthermore, even when we have $c_j = 1$ for all $j \in \mathcal{J}$, there is still room for improvement on the competitive factor.

Our Main Idea. The examples provided in Lemma 5 give a rough idea on the drawbacks of \mathcal{L} , which are twofold: (1) Scheduling jobs on different processors using a global priority queue can easily result in deadline misses. (2) Blindly delaying the jobs can lead to a less energy-efficient schedule.

The former one is more straightforward to deal with. By suitably partitioning the job set, the feasibility can be assured by our assumption on Condition (1). For the latter problem, we introduce the concept of *energy-efficient anchors* for the jobs in order to determine the appropriate timing to begin their execution.

4.1 Our Algorithm

We define some notations to help present our online algorithm \mathcal{S} and the analysis that follows. Let \mathcal{J} be the input set of jobs, and recall that $\mathcal{J}(t)$ is the subset of jobs whose arrival times are smaller than or equal to t .

For any t, t^\dagger with $0 \leq t \leq t^\dagger$, let $\mathbf{Q}(t)$ be the subset of $\mathcal{J}(t)$ that contains the jobs that have not yet finished their execution, and let $\mathbf{Q}(t, t^\dagger)$ be the subset of $\mathbf{Q}(t)$ containing those jobs whose deadlines are smaller than or equal to t^\dagger . Note that, by definition, we have $\mathbf{Q}(t, t^\dagger) \subseteq \mathbf{Q}(t) \subseteq \mathcal{J}(t) \subseteq \mathcal{J}$. For notational brevity, let $c'_j(t)$ denote the remaining execution time of job j at time t , and let $W(t) = \sum_{j \in \mathbf{Q}(t)} c'_j(t)$ and $W(t, t^\dagger) = \sum_{j \in \mathbf{Q}(t, t^\dagger)} c'_j(t)$ denote the total remaining execution time of the jobs in $\mathbf{Q}(t)$ and $\mathbf{Q}(t, t^\dagger)$, respectively. Furthermore, we divide $\mathbf{Q}(t)$ into two subsets according to the arrival times of the jobs. For any t, t^* with $0 \leq t^* \leq t$, let $\mathbf{Q}_{proc}^{t^*}(t)$ be the subset of $\mathbf{Q}(t)$ containing the jobs whose arrival times are less than t^* , and let $\mathbf{Q}_{forth}^{t^*}(t) = \mathbf{Q}(t) \setminus \mathbf{Q}_{proc}^{t^*}(t)$.

Let λ , $0 \leq \lambda \leq 1$, be a constant to be determined later. For each job $j \in \mathcal{J}$, we define a parameter h_j to be $\max\{a_j, d_j - \lambda\mathcal{B}\}$. The value h_j is referred to as the *energy-efficient anchor* for job j .

Let M_1 and M_2 denote the two processors which our algorithm \mathcal{S} will manage. We say that the system is *running*, if at least one processor is executing a job. The system is said to be *off* if all processors are turned off. Otherwise, the system is said to be in *standby*. During the process of job scheduling, our algorithm \mathcal{S} maintains an *urgency flag*, which is initialized to be *false*. At any time t , \mathcal{S} proceeds as follows.

(A) Conditions for switching on the processors:

- (a) **If** the system is *off* and there exists some $j \in \mathbf{Q}(t)$ such that $h_j \geq t$, **then** turn on processor M_1 .
- (b) **If** the *urgency flag* is *false* and there exists some t^\dagger with $t^\dagger > t$ such that $W(t, t^\dagger) \geq t^\dagger - t$, **then**
 - turn on M_1 if it is *off*,
 - turn on M_2 , set t^* to be t , and set the *urgency flag* to be *true*.

(B) To handle the job scheduling:

- (a) **If** the *urgency flag* is *true*, **then** use the earliest-deadline-first principle to schedule jobs from $\mathbf{Q}_{proc}^{t^*}(t)$ on M_1 and jobs from $\mathbf{Q}_{forth}^{t^*}(t)$ on M_2 .
- (b) **If** the *urgency flag* is *false* and the system is not *off*, **then** use the EDF principle to schedule jobs from $\mathbf{Q}(t)$ on the processor that is *on*.

(C) Conditions for turning off the processors:

- (a) **If** the *urgency flag* is *true* and $\mathbf{Q}_{proc}^{t^*}(t)$ becomes empty, **then** turn off M_1 and set the *urgency flag* to be *false*.
- (b) **If** the *urgency flag* is *false*, the system is *standby*, and $t - t_1 \geq \mathcal{B}$, where t_1 is the time processor M_1 was turned on, **then** turn off all processors.

Note that, M_1 and M_2 can both be on only when the *urgency flag* is *true*.

4.2 The Analysis

Let $I_1 = (\ell_1, r_1), I_2 = (\ell_2, r_2), \dots, I_\kappa = (\ell_\kappa, r_\kappa)$, where $r_i < \ell_j$ for $1 \leq i < j \leq \kappa$, be the set of time intervals during which the system is either *running* or *in standby*. We also refer these intervals to as the *awaken-intervals*. For ease of presentation, let $I_0 = (0, 0)$ be a dummy *awaken-interval*.

Lemma 6. Provided that Condition (1) from Lemma 1 holds for the input jobs, algorithm \mathcal{S} always produces a feasible schedule.

Below, we bound the competitive factor of \mathcal{S} . Let \mathcal{P}_i be the number of times \mathcal{S} turns on a processor in I_i . We have the following lemma regarding \mathcal{P}_i .

Lemma 7. For each i , $1 \leq i \leq \kappa$, we have

- $\mathcal{P}_i \leq 2$.
- If $\mathcal{P}_i = 2$, then the amount of workload that arrives after r_{i-1} and has to be done before r_i is at least $\lambda\mathcal{B}$.

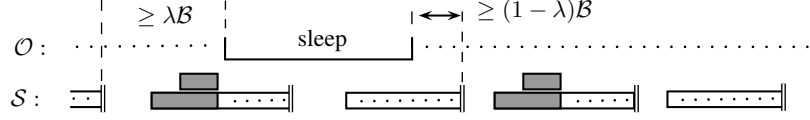


Figure 1: The relative positions between sleep intervals of \mathcal{O} and awoken-intervals of \mathcal{S} .

Let \mathbf{J}_i denote the set of the jobs executed in awoken-interval I_i in \mathcal{S} . As a consequence to Lemma 7, we have $\sum_{j \in \mathbf{J}_i} c_j \geq \lambda \mathcal{B}$ for each I_i with $\mathcal{P}_i = 2$, $1 \leq i \leq \kappa$. Let \mathcal{O} be an optimal offline uni-processor schedule. Below, we relate our online schedule to the optimal offline schedule \mathcal{O} . Suppose that $U_i = (\ell_{i,opt}, r_{i,opt})$, $1 \leq i \leq m$, is the set of intervals for which \mathcal{O} keeps the system off. We also refer these intervals to as the *sleep intervals* of \mathcal{O} .

Lemma 8. For any i , $1 \leq i \leq \kappa$, if both r_{i-1} and ℓ_i are contained in some sleep interval $U_u = (\ell_{u,opt}, r_{u,opt})$ of \mathcal{O} , then $\mathcal{P}_i = 1$ and $r_i - r_{u,opt} \geq (1 - \lambda)\mathcal{B}$.

Lemma 9. Each sleep interval of \mathcal{O} intersects with at most two awoken-intervals of \mathcal{S} . In particular, each of U_1 and U_m intersects with at most one.

The above two lemmas characterize the relative positions between the sleep intervals of \mathcal{O} and the awoken intervals they intersect with. See also Fig. 1 for an illustration. For simplicity, we implicitly concatenate U_1 and U_m , i.e., we assume that the left end of U_1 is connected to the right end of U_m for the rest of this section.

Below we upper-bound the energy consumption of \mathcal{S} by the energy consumption of \mathcal{O} . In order to help present the analysis, we define the following notations for the energy consumption of the system in \mathcal{S} and in \mathcal{O} : (a) $\mathcal{E}_{w,\sigma}$: energy for waking up and standby, (b) \mathcal{E}_w : energy for waking up, (c) \mathcal{E}_b : energy for executing jobs, and (d) $\mathcal{E}_{b,\sigma}$: energy for executing jobs and standby.

We use $\mathcal{E}_{w,\sigma}(\mathcal{S}, t_1, t_2)$, and $\mathcal{E}_b(\mathcal{S}, t_1, t_2)$ to denote the corresponding energy \mathcal{S} consumes between time t_1 and t_2 . Furthermore, $\mathcal{E}_{b,\sigma}(\mathcal{O}, t_1, t_2)$, and $\mathcal{E}_w(\mathcal{O}, t_1, t_2)$ are the corresponding energies \mathcal{O} consumes between time t_1 and t_2 .

Lemma 10. For each awoken-interval I_i , $1 \leq i \leq \kappa$ and $\lambda \leq 1$,

$$\mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \leq \psi_\sigma \left(\sum_{j \in \mathbf{J}_i} c_j \right) + (3 - \lambda)E_w. \quad (2)$$

$$\text{Furthermore, if } \mathcal{P}_i = 1, \text{ then } \mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \leq 2E_w. \quad (3)$$

By combining all the above analysis in Lemmas 7, 8, 9, and 10 for the properties between the optimal offline schedule and the schedule derived from \mathcal{S} , we prove an upper bound on the energy consumption $\mathcal{E}_{w,\sigma}(\mathcal{S}, 0, \infty)$ in the following lemma.

Lemma 11. When $2 - \sqrt{3} \leq \lambda \leq 1$, we have

$$\mathcal{E}_{w,\sigma}(\mathcal{S}, 0, \infty) \leq (4 - \lambda) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, 0, \infty) + 4 \cdot \mathcal{E}_w(\mathcal{O}, 0, \infty).$$

Proof. Consider each of the following three exclusive cases of an awoken-interval I_i , for $1 \leq i \leq \kappa$.

Case 1: Interval I_i does not intersect with any sleep interval of \mathcal{O} . Therefore, in the optimal schedule \mathcal{O} , the processor is not off during time interval (ℓ_i, r_i) . By our algorithm design, the length of an awoken-interval is at least \mathcal{B} . Hence $\mathcal{E}_{b,\sigma}(\mathcal{O}, \ell_i, r_i) \geq \mathcal{B} \cdot \psi_\sigma = E_w$. By Lemma 10, when $\lambda \leq 1$, we have

$$\mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \leq \psi_\sigma \cdot \sum_{j \in \mathbf{J}_i} c_j + (3 - \lambda) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, \ell_i, r_i). \quad (4)$$

Case 2: Interval I_i intersects with a sleep interval $U_u = (\ell_{u,opt}, r_{u,opt})$ and r_i is contained in interval U_u . We have two subcases: (a) If $\mathcal{P}_i = 1$, by Eq. (3) in Lemma 10, we have $\mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \leq 2E_w = 2\mathcal{E}_w(\mathcal{O}, \ell_{u,opt}, r_{u,opt})$, in which the equality comes by the definition of a sleep interval. (b) If $\mathcal{P}_i = 2$, to provide an upper bound to Eq. (2) in Lemma 10, we use the energy consumption $\mathcal{E}_w(\mathcal{O}, \ell_{u,opt}, r_{u,opt})$ in the optimal schedule to bound $2E_w$, and the energy consumption $\mathcal{E}_{b,\sigma}(\mathcal{O}, r_{i-1}, \ell_{u,opt})$ in the optimal schedule to bound $(1 - \lambda)E_w$. We know that, when $2 - \sqrt{3} \leq \lambda \leq 1$, we have

$$1 - \lambda \leq \lambda(3 - \lambda). \quad (5)$$

By Lemma 7, in \mathcal{O} , the workload that has to be done between r_{i-1} and $\ell_{u,opt}$ is lower-bounded by $\lambda\mathcal{B}$. Since $r_i \geq \ell_{u,opt}$, the workload that has to be done between r_{i-1} and r_i is also lower-bounded by $\lambda\mathcal{B}$, and hence,

$$\mathcal{E}_{b,\sigma}(\mathcal{O}, r_{i-1}, r_i) \geq \psi_\sigma \cdot \lambda\mathcal{B} = \lambda E_w, \quad (6)$$

Hence, by Eq. (5) and Eq. (6), we have $(1 - \lambda)E_w \leq (3 - \lambda) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, r_{i-1}, r_i)$. By rephrasing Eq. (2) in Lemma 10, for both subcases, we have

$$\begin{aligned} \mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) &\leq \psi_\sigma \sum_{j \in \mathbf{J}_i} c_j + (1 - \lambda) \cdot E_w + 2 \cdot E_w \\ &\leq \psi_\sigma \cdot \sum_{j \in \mathbf{J}_i} c_j + (3 - \lambda) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, r_{i-1}, r_i) + 2 \cdot \mathcal{E}_w(\mathcal{O}, \ell_{u,opt}, r_{u,opt}). \end{aligned} \quad (7)$$

Case 3: I_i intersects exactly with one sleep interval $U_u = (\ell_{u,opt}, r_{u,opt})$ and ℓ_i is contained in interval U_u , then, by Lemma 8, we have $\mathcal{P}_i = 1$ and therefore

$$\mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \leq 2 \cdot E_w \leq 2 \cdot \mathcal{E}_w(\mathcal{O}, \ell_{u,opt}, r_{u,opt}). \quad (8)$$

Provided the individual upper bounds for each awoken interval in Eq. (4), Eq. (7), and Eq. (8), we combine them to get an overall upper bound for $\mathcal{E}_{w,\sigma}(\mathcal{S}, 0, \infty)$, which is the summation over each awoken interval, $\sum_{1 \leq i \leq \kappa} \mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i)$.

First, since the awoken intervals are mutually disjoint, the item $\mathcal{E}_{b,\sigma}(\mathcal{O}, r_{i-1}, r_i)$ is counted at most once for each i , $1 \leq i \leq \kappa$. Note that, $\sum_{1 \leq i \leq \kappa} \mathcal{E}_{b,\sigma}(\mathcal{O}, r_{i-1}, r_i) \leq \mathcal{E}_{b,\sigma}(\mathcal{O}, 0, \infty)$. Second, by Lemma 9, a sleep interval U_u intersects with at most two awoken-intervals of \mathcal{S} . Hence, $\mathcal{E}_w(\mathcal{O}, \ell_u, r_u)$ is counted at most twice. Therefore, when $2 - \sqrt{3} \leq \lambda \leq 1$, we have

$$\begin{aligned} \mathcal{E}_{w,\sigma}(\mathcal{S}, 0, \infty) &= \sum_{1 \leq i \leq \kappa} \mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \\ &\leq \sum_{1 \leq i \leq \kappa} \left(\psi_\sigma \cdot \sum_{j \in \mathbf{J}_i} c_j \right) + (3 - \lambda) \cdot \sum_{1 \leq i \leq \kappa} \mathcal{E}_{b,\sigma}(\mathcal{O}, r_{i-1}, r_i) \\ &\quad + 4 \cdot \sum_{1 \leq u \leq m} \mathcal{E}_w(\mathcal{O}, \ell_{u,opt}, r_{u,opt}) \\ &\leq (4 - \lambda) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, 0, \infty) + 4 \cdot \mathcal{E}_w(\mathcal{O}, 0, \infty), \end{aligned}$$

where the last inequality comes from the assumption that $\psi_\sigma \leq \psi_b$. □ □

Theorem 12. By setting $\lambda = 1$, algorithm \mathcal{S} computes a 4-competitive schedule for any given set of jobs satisfying Condition (1) for the online power-minimizing scheduling problem.

Proof. The feasibility of \mathcal{S} is guaranteed by Lemma 6. For the competitive factor, let $\mathcal{E}(\mathcal{S})$ and $\mathcal{E}(\mathcal{O})$ denote the total energy consumption for schedule \mathcal{S} and \mathcal{O} , respectively. First, we have $\mathcal{E}_b(\mathcal{S}, 0, \infty) \leq \mathcal{E}_{b,\sigma}(\mathcal{O}, 0, \infty)$, since the workload to execute is the same in both \mathcal{S} and \mathcal{O} . Together with Lemma 11, we have $\mathcal{E}(\mathcal{S}) = \mathcal{E}_b(\mathcal{S}, 0, \infty) + \mathcal{E}_{w,\sigma}(\mathcal{S}, 0, \infty) \leq (5 - \lambda) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, 0, \infty) + 4 \cdot \mathcal{E}_w(\mathcal{O}, 0, \infty) \leq (5 - \lambda) \cdot \mathcal{E}(\mathcal{O})$, provided that $2 - \sqrt{3} \leq \lambda \leq 1$. By choosing λ to be 1, we have $\mathcal{E}(\mathcal{S}) \leq 4 \cdot \mathcal{E}(\mathcal{O})$ as claimed. □ □

4.3 Jobs with Unit Execution Time

We show that, when the execution times of the jobs are unit, we can benefit even more from the energy-efficient anchor with a properly chosen parameter $\lambda = 4 - \sqrt{10}$. The major difference is that, when the system is in urgency while $\mathbf{Q}_{forth}^{t^*}(t)$ is empty, i.e., processor M_2 is in standby, we use a global earliest-deadline-first scheduling by executing two jobs on M_1 and M_2 instead of keeping one processor in standby, which improves resource utilization. Let \mathcal{S}^\dagger denote the modified algorithm.

As a result, both processors will keep running until the global ready queue is empty. Hence, we know that, when $\mathcal{P}_i = 2$ for an awaken-interval I_i of \mathcal{S}^\dagger , the total standby time in I_i is at most $(1 - \frac{\lambda}{2})\mathcal{B}$. The following two lemmas are the updated versions of Lemma 10 and Lemma 11, respectively.

Lemma 13. When the jobs have unit execution times, for each awaken-interval I_i , $1 \leq i \leq k$, we have $\mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, \ell_i, r_i) \leq (3 - \frac{\lambda}{2})E_w$.

Lemma 14. When $7 - \sqrt{41} \leq \lambda \leq 4 - \sqrt{10}$, we have

$$\mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, 0, \infty) \leq \left(3 - \frac{\lambda}{2}\right) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, 0, \infty) + \left(4 - \frac{\lambda}{2}\right) \cdot \mathcal{E}_w(\mathcal{O}, 0, \infty).$$

By Lemma 14 with λ chosen to be $4 - \sqrt{10}$, we get $\mathcal{E}(\mathcal{S}^\dagger) \leq (4 - \frac{\lambda}{2}) \cdot \mathcal{E}(\mathcal{O}) < 3.59 \cdot \mathcal{E}(\mathcal{O})$. We have the following theorem.

Theorem 15. By setting $\lambda = 4 - \sqrt{10}$, algorithm \mathcal{S}^\dagger computes a 3.59-competitive schedule for any given set of jobs with execution time that satisfies Condition (1) for the online power-minimizing scheduling problem.

4.4 Trading the Number of Processors with the Energy-Efficiency

We have shown how a stream of jobs satisfying Condition (1) can be scheduled online to obtain a 4-competitive schedule which uses at most two processors. By collecting the delayed jobs and bundling their execution, we can generalize the algorithm to a prescribed collection of job streams, each of which satisfies Condition (1), to allow a trade-off between the number of processors we use and the energy-efficiency of the resulting schedule.

Let $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_k$ be the given collection of job streams, where \mathcal{J}_i satisfies Condition (1) for all $1 \leq i \leq k$. Below we show that, for any fixed h , $h > k$, how a $(4 \cdot \max\{\lceil \frac{k}{h-k} \rceil, 1\})$ -competitive schedule which uses at most h processors can be obtained. First, if $h \geq 2k$, then we simply apply the algorithm \mathcal{S} on every pair of the streams, i.e., on \mathcal{J}_{2i} and \mathcal{J}_{2i+1} , for all $1 \leq i \leq \frac{k}{2}$, and we get a 4-competitive schedule. For the case $k < h < 2k$, we divide $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_k$ into $h - k$ subsets such that each subset gets at most $\lceil k/(h - k) \rceil$ streams. The h processors are allocated in the following way. Each stream of jobs gets one processor, and the remaining $h - k$ processors are equally allocated to each of the $h - k$ subsets. Then, the problem is reduced to the remaining case, $h = k + 1$.

Below we describe how the case $h = k + 1$ is handled. Let M_0, M_1, \dots, M_k denote the $k + 1$ processors to be managed, and let $\mathbf{Q}_0, \mathbf{Q}_1, \dots, \mathbf{Q}_k$ be the corresponding subset of jobs that are scheduled to be executed on each processor. We use the parameter $\lambda = 1$ to set the energy-efficient anchor for each job that arrives. Let $W_0(t, t^\dagger) = \sum_{j \in \mathbf{Q}_0, d_j \leq t^\dagger} c'_j(t)$ denote the total remaining execution time of the jobs in \mathbf{Q}_0 whose deadlines are less than or equal to t^\dagger , for any time t and any t^\dagger with $t^\dagger \geq t$. The algorithm works as follows. When a job $j \in \mathcal{J}_i$, $1 \leq i \leq k$, arrives, if M_i is on, then we add j to \mathbf{Q}_i . Otherwise, we further check whether $W_0(t, t^\dagger) + c_j \leq t^\dagger - t$ holds for all $t^\dagger \geq d_j$. If it does, then j is added to \mathbf{Q}_0 . Otherwise, we add j to \mathbf{Q}_i and turn on the processors M_i and M_0 (if M_0 is off). At any time t , we have the following cases to consider.

- First, regarding the further conditions to turn on M_0 , if M_0 is off, and if the energy-efficient anchor of some job in \mathbf{Q}_0 is met or there exists some $t^\dagger \geq t$ such that $W_0(t, t^\dagger) \geq t^\dagger - t$, then we turn on M_0 .
- Second, regarding the scheduling, for each $0 \leq i \leq k$ such that M_i is on, we use the earliest-deadline-first principle to schedule the jobs of \mathbf{Q}_i on M_i .
- Finally, regarding the conditions for turning off the processor, when \mathbf{Q}_0 becomes empty and M_0 has been turned on for at least \mathcal{B} amount of time, we turn off M_0 . For $1 \leq i \leq k$, if M_i is on, \mathbf{Q}_i becomes empty, and M_0 is off, then we turn off M_i .

Theorem 16. Given a collection of k job sets, each satisfying Condition (1), we can compute a $\left(4 \cdot \max \left\{ \left\lceil \frac{k}{h-k} \right\rceil, 1 \right\} \right)$ -competitive schedule that uses at most h processors, where $h > k$, for the online power-minimizing scheduling problem.

5 Conclusion

We conclude with a brief overview on future research directions. With the advances of technology, racing-to-idle has become an important scheduling model for energy efficiency. For online settings, it still remains open how a balance between (1) the extent to which the executions of the jobs should be postponed, and (2) the extent to which the system should speed up for impulse arrivals, can be reached or even be traded with other parameters. We believe that this will be a very interesting research direction to explore.

References

- [1] Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. In *SODA*, pages 1266–1285, 2012.
- [2] John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal power-down strategies. In *FOCS*, pages 530–539, 2004.
- [3] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: A polynomial time algorithm for offline dynamic power management. In *SODA*, pages 364–367, 2006.
- [4] Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial time algorithms for minimum energy scheduling. In *ESA*, pages 136–150, 2007.
- [5] Jian-Jia Chen and Tei-Wei Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. In *LCTES*, 2006.
- [6] Jian-Jia Chen and Tei-Wei Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *ICCAD*, 2007.
- [7] Houssine Chetto and Maryline Silly-Chetto. Scheduling periodic and sporadic tasks in a real-time system. *Inf. Process. Lett.*, 30(4):177–184, 1989.
- [8] Julia Chuzhoy and Paolo Codenotti. Resource minimization job scheduling. In *APPROX’09/RANDOM’09*, pages 70–83, Berlin, Heidelberg, 2009. Springer-Verlag.
- [9] Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *FOCS’04*, 2004.
- [10] M. Cieliebak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *IFIP*, pages 209–222. 2004.

- [11] Michael L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.
- [12] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.
- [13] Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C. Buttazzo. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems*, 47(2):163–193, 2011.
- [14] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. In *SODA*, pages 37–46, 2003.
- [15] Sandy Irani, Sandeep K. Shukla, and Rajesh K. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embedded Comput. Syst.*, 2(3):325–346, 2003.
- [16] Mong-Jen Kao, Jian-Jia Chen, Ignaz Rutter, and Dorothea Wagner. Competitive design and analysis for machine-minimizing job scheduling problem. In *ISAAC*, pages 75–84, 2012.
- [17] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, 1994.
- [18] Yann-Hang Lee, Krishna P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *ECRTS*, pages 105–112, 2003.
- [19] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.

A Problem Lower Bound

Lemma 2. *If the first job released by \mathcal{A} is scheduled at time t with $0 \leq t \leq (\frac{1}{2} - x)\mathcal{B}$, then we have*

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq 2 + \frac{1}{2}x - O\left(\frac{1}{k}\right) \quad \text{for any } 0 \leq x \leq \frac{2}{5}.$$

Proof of Lemma 2. Let m be the number of jobs \mathcal{A} releases, including the first job at time 0. Suppose that these m jobs are indexed in an ascending order according to their arrival time. By definition $a_1 = 0$ and $a_m \leq \frac{3}{2}\mathcal{B}$.

Hence, the online algorithm Π turns on a processor at least m times, in which the energy consumption for switching on is at least $m \cdot E_w$. Moreover, except for $(m-1)\epsilon_0$ units of time when no processor is on, Π keeps a processor on for $(\frac{3\mathcal{B}}{2} - t) - (m-1)\epsilon_0$ time units in the time interval $[t, \frac{3\mathcal{B}}{2}]$. Therefore $\mathcal{E}(\Pi)$ is at least $mE_w + (\frac{3\mathcal{B}}{2} - t)\psi - (m-1)\epsilon_0\psi$, which is at least $mE_w + (1+x)\mathcal{B}\psi - (m-1)\psi$.

Then, we give the upper bound for the energy consumption of an optimal offline schedule by constructing a specific feasible offline schedule for these m jobs. Note that, the job \mathcal{A} releases at time 0 is flexible and can be scheduled at any time between t and $\mathcal{B} - 1$. Specifically, we consider 4 cases, depending on the value of m .

When $m = 1$, we have only one job to execute and one feasible schedule is to turn on the processor at any time $t < \mathcal{B} - 1$ followed by immediately turning off. Therefore, for $m = 1$, $\mathcal{E}(\mathcal{O}) \leq E_w + \psi$, and

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \frac{k(2+x)}{k+1} = (2+x) - O\left(\frac{1}{k}\right).$$

When $m = 2$, there are two subcases. (1) If $a_2 \leq \mathcal{B}$, then the two jobs can be scheduled to run consecutively and the energy consumption is $E_w + (1 + \epsilon_0)\psi$. (2) If $a_2 > \mathcal{B}$, the first job can be scheduled to run at time $\mathcal{B} - 1$, and the processor idles from \mathcal{B} to a_2 and executes the second job at time a_2 . The energy consumption is $E_w + (a_2 - \mathcal{B} + (1 + \epsilon_0))\psi$, which is maximized when $a_2 = \frac{3}{2}\mathcal{B}$. Therefore, when $m = 2$, for both subcases we have $\mathcal{E}(\mathcal{O}) \leq E_w + (\frac{\mathcal{B}}{2} + 2)\psi$, and

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \frac{k(3+x) - 1}{\frac{3k}{2} + 2} = 2 + \frac{2}{3}x - O\left(\frac{1}{k}\right).$$

When $m = 3$, there are two subcases as well. (1) If $a_3 - a_2 > \mathcal{B}$, then $a_2 < \mathcal{B}$, and the first two jobs can be scheduled to run consecutively. The processor is off from $a_2 + \epsilon_0$ to a_3 . The energy consumption for this subcase is $2E_w + (1 + 2\epsilon_0)\psi$. (2) If $a_3 - a_2 \leq \mathcal{B}$, we schedule the first job at any time between $\frac{1}{2}\mathcal{B}$ and $\mathcal{B} - 1$. Disregarding $a_2 \leq \mathcal{B}$ or not, the energy consumption for this subcase is at most $E_w + (\mathcal{B} + \epsilon_0)\psi$. Therefore, when $m = 3$, we have $\mathcal{E}(\mathcal{O}) \leq 2E_w + 3\psi$, and

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \frac{k(4+x) - 2}{2k + 3} = 2 + \frac{1}{2}x - O\left(\frac{1}{k}\right).$$

When $m \geq 4$, we keep the processor on from time t to time $\frac{3\mathcal{B}}{2} + \epsilon_0$. The energy consumption for the optimal offline schedule is at most $E_w + (\frac{3\mathcal{B}}{2} - t + 1)\psi$, and

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \frac{4k + (\frac{3k}{2} - t) - 3}{k + \frac{3k}{2} - t + 1} = \frac{11k - 2t - 6}{5k - 2t + 2}.$$

Since $\mathcal{E}(\Pi) > \mathcal{E}(\mathcal{O}) \geq 0$ and $t \geq 0$, the above fraction achieves its minimum at $t = 0$. Hence, when $m = 4$, we have

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \frac{11k - 6}{5k + 2} = \frac{11}{5} - O\left(\frac{1}{k}\right).$$

Combining the above four cases, we get

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \min \left\{ 2 + x, 2 + \frac{2}{3}x, 2 + \frac{1}{2}x, \frac{11}{5} \right\} - O\left(\frac{1}{k}\right) = 2 + \frac{1}{2}x - O\left(\frac{1}{k}\right),$$

when k is large enough and $0 \leq x \leq \frac{2}{5}$. \square

Lemma 3. *If the first job released by the adversary is not executed until time $(\frac{1}{2} - x)\mathcal{B}$, then for any $x, \eta, \chi \geq 0$ we have*

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \min \left\{ \frac{3 + x + \eta}{\frac{3}{2} + x}, \frac{4 + x + \eta + \chi}{2 + x + \eta}, \frac{5 + x + \eta + \chi}{2 + x + \eta + \chi} \right\} - O\left(\frac{1}{k}\right).$$

Proof of Lemma 3. By our design, \mathcal{A} releases exactly $(\frac{1}{2} + x)\mathcal{B}$ jobs before time \mathcal{B} , forcing Π to use at least two processors to have a feasible schedule. Let m_1 be the number of the urgent unit jobs that \mathcal{A} releases between time \mathcal{B} and time $(\frac{3}{2} + \eta)\mathcal{B}$. If $m_1 = 0$, then Π keeps at least one processor in the standby mode till time $(\frac{3}{2} + \eta)\mathcal{B}$, and the energy consumption is $\mathcal{E}(\Pi) \geq 2 \cdot E_w + (\frac{1}{2} + x)\mathcal{B} \cdot \psi + (\frac{1}{2} + \eta)\mathcal{B} \cdot \psi$. A feasible offline schedule can execute all these jobs consecutively on a processor from time $(\frac{1}{2} - x)\mathcal{B}$ to \mathcal{B} , with energy consumption equal to $E_w + (\frac{1}{2} + x)\mathcal{B} \cdot \psi$. Therefore, for $m_1 = 0$, we know that $\mathcal{E}(\mathcal{O}) \leq E_w + (\frac{1}{2} + x)\mathcal{B} \cdot \psi$, which also implies

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \frac{3 + x + \eta}{\frac{3}{2} + x}. \quad (9)$$

For the other case with $m_1 \geq 1$, the adversary \mathcal{A} monitors the behavior of Π till time $(\frac{3}{2} + \eta + \chi)\mathcal{B}$. Let m_2 be the number of the urgent unit jobs \mathcal{A} releases after time $(\frac{3}{2} + \eta)\mathcal{B}$. Then by an analogous argument, we have $\mathcal{E}(\Pi) \geq (2 + m_1 + m_2) \cdot E_w + (\frac{1}{2} + x)\mathcal{B} \cdot \psi + (\frac{1}{2} + \eta + \chi)\mathcal{B} \cdot \psi - (m_1 + m_2)\psi$. Note that this function is minimized when m_1 and m_2 achieve their minimums.

A feasible offline schedule on a processor for $m_1 \geq 1$ is constructed by turning on the processor at time $(\frac{1}{2} - x)\mathcal{B}$ and executing the coming jobs. If $m_2 = 0$, then we turn off the processor at time $(\frac{3}{2} + \eta)\mathcal{B}$, and, hence, $\mathcal{E}(\mathcal{O}) \leq E_w + (1 + x + \eta)\mathcal{B}\psi$. Otherwise, we turn the processor off at time $(\frac{3}{2} + \eta + \chi)\mathcal{B}$, and, hence, $\mathcal{E}(\mathcal{O}) \leq E_w + (1 + x + \eta + \chi)\mathcal{B}\psi$. Therefore,

$$\begin{aligned} \frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} &\geq \min \left\{ \frac{(3 + m_1 + x + \eta + \chi)k - m_1}{(2 + x + \eta) \cdot k}, \frac{(3 + m_1 + m_2 + x + \eta + \chi)k - (m_1 + m_2)}{(2 + x + \eta + \chi) \cdot k} \right\} \\ &\geq \min \left\{ \frac{4 + x + \eta + \chi}{2 + x + \eta}, \frac{5 + x + \eta + \chi}{2 + x + \eta + \chi} \right\} - O\left(\frac{1}{k}\right) \end{aligned} \quad (10)$$

By Eqs. (9) and (10), the lemma is proved. \square

Theorem 4. *The competitive factor of any online algorithm Π for the online power-minimizing scheduling problem is at least 2.06.*

Proof of Theorem 4. By Lemmas 2 and 3, we know that

$$\frac{\mathcal{E}(\Pi)}{\mathcal{E}(\mathcal{O})} \geq \sup_{\substack{0 \leq x \leq 2/5 \\ 0 \leq \eta, \chi}} \min \left\{ \frac{4 + x}{2}, \frac{3 + x + \eta}{\frac{3}{2} + x}, \frac{4 + x + \eta + \chi}{2 + x + \eta}, \frac{5 + x + \eta + \chi}{2 + x + \eta + \chi} \right\} - O\left(\frac{1}{k}\right).$$

To get an asymptotic sup-min of the four items inside the brace, observe that, as χ increases, the third item increases while the fourth item decreases. Therefore, the value of χ for which the last two items achieve their max-min can be solved. In this way, we can solve η and x to get the asymptotic lower bound for $\mathcal{E}(\Pi)/\mathcal{E}(\mathcal{O})$.

By choosing x , η , and χ to be 0.1218, 0.2206, and 0.4852, respectively, we get $\mathcal{E}(\Pi) \geq 2.06 \cdot \mathcal{E}(\mathcal{O})$ when $k \gg 1$. \square

B Online Scheduling

B.1 Common Approaches and Bad Examples

Lemma 5 . *The competitive factor of \mathcal{L} can be arbitrarily bad. Furthermore, even when $c_j = 1$ for all $j \in \mathcal{J}$, the competitive factor of \mathcal{L} is still at least 6.*

Proof of Lemma 5. For the first part of this lemma, consider the following job set, \mathbf{J}_{edf} . Let k be a non-negative integer. At time 0, a job $j_1 = (0, 4k, 3k)$ arrives and is delayed. The system remains in standby until time k , at which $k - 1$ unit jobs arrive to the system with deadline $4k - 1$. As a result, j_1 will miss its deadline unless we turn on k processors. This shows that, the number of processors \mathcal{L} will use for \mathbf{J}_{edf} can be arbitrarily large, whereas the optimal schedule uses only one processor, and the competitive factor can therefore be arbitrarily large.

For the second part of this lemma, we define another job set \mathbf{J} . Let $n > 0$ be an even integer. The job set \mathbf{J} consists of $\frac{3}{2}n$ jobs: $j_1, j_2, \dots, j_n, j'_1, j'_2, \dots, j'_{n/2}$, where

$$\begin{cases} j_i = ((i-1)\mathcal{B} + 2i, i\mathcal{B} + 2i + 1), & \text{for each } 1 \leq i \leq n, \text{ and} \\ j'_k = (2k\mathcal{B} + 4k, 2k\mathcal{B} + 4k + 1), & \text{for each } 1 \leq k \leq \frac{n}{2}. \end{cases}$$

Let $\mathcal{E}(\mathcal{L})$ and $\mathcal{E}(\mathcal{O})$ be the energy consumed by \mathcal{L} and the offline optimal schedule for the input instance \mathbf{J} , respectively. We prove this theorem by showing that $\mathcal{E}(\mathcal{L})$ is asymptotically no less than $6 \cdot \mathcal{E}(\mathcal{O})$.

First we consider the behavior of the algorithm \mathcal{L} . At the beginning, \mathcal{L} postpones the execution of j_1 until time $\mathcal{B} + 2$, for which it turns on a processor to execute j_1 , followed by staying in standby for \mathcal{B} units of time until it turns off the processor at time $2\mathcal{B} + 3$. At time $2\mathcal{B} + 4$ when \mathcal{L} will schedule j_2 to execute, an urgent job j'_1 arrives. As a result, \mathcal{L} has to turn on another processor for it. Then \mathcal{L} keeps both processors in standby followed by turning them off at time $3\mathcal{B} + 5$. The same pattern is repeated for the remaining $\frac{3n}{2} - 3$ jobs. See also Fig 2 for an illustration. For each repetition of the pattern, say, for jobs j_{2k-1}, j_{2k} , and j'_k , $1 \leq k \leq \frac{n}{2}$, the energy consumption of \mathcal{L} is at least $3 \cdot E_w + 3\mathcal{B} \cdot \psi_\sigma = 6E_w$. Therefore $\mathcal{E}(\mathcal{L}) \geq \frac{n}{2} \cdot 6E_w = 3nE_w$.

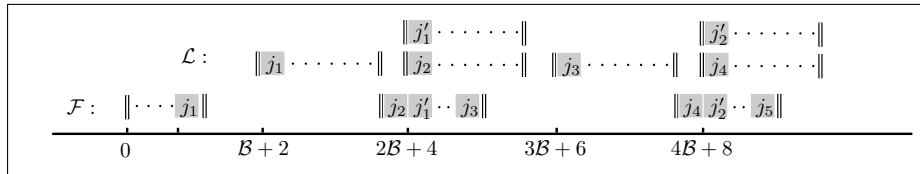


Figure 2: Dotted horizontal lines represent the periods for which the processor is in the standby mode, and double vertical lines represent the turn-ons and turn-offs.

A feasible schedule for \mathbf{J} , denoted by \mathcal{F} , can be constructed as follows. For each $1 \leq k \leq \frac{n}{2}$, we turn on a processor at time $\max\{0, (2k-2)\mathcal{B} + 4k - 5\}$ and turn off the processor at time $(2k-2)\mathcal{B} + 4k - 1$. We schedule j_{2k-1} at the time it arrives, which is at $(2k-2)\mathcal{B} + 4k - 2$. Then, j_{2k} and j'_k are scheduled at $2k\mathcal{B} + 4k - 1$ and $2k\mathcal{B} + 4k$, respectively. The energy consumption of this schedule, which is also an upper-bound of $\mathcal{E}(\mathcal{O})$, is at most $(\frac{n}{2} + 1)(E_w + 4\psi_b)$. Therefore,

$$\frac{\mathcal{E}(\mathcal{L})}{\mathcal{E}(\mathcal{O})} \geq \frac{3nE_w}{(\frac{n}{2} + 1)(E_w + 4\psi_b)} = 6 - O\left(\frac{1}{E_w + \psi_b}\right) - O\left(\frac{1}{n}\right),$$

which converges to 6 when $E_w \gg \psi_b \geq 1$ and $n \gg 1$. □ □

B.2 The Analysis

Lemma 6. *Provided that Condition (1) from Lemma 1 holds for the input jobs, algorithm \mathcal{S} always produces a feasible schedule.*

Proof of Lemma 6. Consider any awaken-interval I_i , where $1 \leq i \leq \kappa$. If the urgency flag is never set true within I_i , then we know that only M_1 is turned on during I_i , and we have $W(\ell_i, t) < t - \ell_i$ for all $t > \ell_i$ at time ℓ_i when processor M_1 is turned on. Moreover, by the design of \mathcal{S} , once M_1 is on, it keeps executing jobs using the EDF principle until the queue $\mathbf{Q}(t)$ becomes empty. Therefore by Lemma 1, the schedule \mathcal{S} produces during I_i is feasible.

On the other hand, if the urgency flag is set true at some moment t^* between ℓ_i and r_i , then by a similar argument, we know that the schedules \mathcal{S} produces for the subsets $\mathbf{Q}_{proc}^{t^*}(t)$ and $\mathbf{Q}_{forth}^{t^*}(t)$ are feasible on M_1 and M_2 , respectively, for all $\ell_i \leq t \leq r_i$. \square \square

Lemma 7. *For each i , $1 \leq i \leq \kappa$, we have*

- $\mathcal{P}_i \leq 2$.
- If $\mathcal{P}_i = 2$, then the amount of workload that arrives after r_{i-1} and has to be done before r_i is at least $\lambda\mathcal{B}$.

Proof of Lemma 7. If the urgency flag is never set true in I_i , then $\mathcal{P}_i = 1$. Otherwise, suppose the system enters and leaves urgency at time t^* and t^\dagger , respectively. By our scheme, $\mathbf{Q}_{proc}^{t^*}(t^\dagger)$ is empty, and $W(t^\dagger, t) \leq t - t^\dagger$ for all $t \geq t^\dagger$ by our assumption on Condition (1). Therefore \mathcal{S} will not turn on M_1 again before I_i ends. This shows $\mathcal{P}_i \leq 2$.

Suppose that \mathcal{S} turns on M_2 at time t^* , and t^\dagger is the corresponding moment for which $W(t^*, t^\dagger) > t^\dagger - t^*$. Then by the design of \mathcal{S} , we know that the ready queue is never empty between time ℓ_i and t^* , for otherwise it will contradict Condition (1) since $W(t^*, t^\dagger) > t^\dagger - t^*$. Therefore the system stays in the running mode and never enters the standby mode between time ℓ_i and t^* .

Since we use EDF principle, the workload that has to be done before t^\dagger is hence strictly greater than $t^\dagger - \ell_i$. Namely, among these jobs, at least one, say, j , has arrival time earlier than ℓ_i . That is, $r_{i-1} \leq a_j < \ell_i$. Therefore $d_j \geq \ell_i + \lambda\mathcal{B}$; otherwise \mathcal{S} would have turned on M_1 earlier than ℓ_i when the energy-efficient anchor of j is met. Therefore the total workload that is done within I_i is at least $t^\dagger - \ell_i \geq d_j - \ell_i \geq \lambda\mathcal{B}$. \square \square

Lemma 8. *For any i , $1 \leq i \leq \kappa$, if both r_{i-1} and ℓ_i are contained in some sleep interval $U_u = (\ell_{u,opt}, r_{u,opt})$ of \mathcal{O} , then $\mathcal{P}_i = 1$ and $r_i - r_{u,opt} \geq (1 - \lambda)\mathcal{B}$.*

Proof of Lemma 8. Since I_{i-1} ends when the ready queue is empty, the arrival time of each job in \mathbf{J}_i is later than r_{i-1} . Therefore $\mathcal{P}_i = 1$ as \mathcal{O} is a feasible uni-processor schedule which turns on the system to process jobs no earlier than ℓ_i .

Let j be the job scheduled to execute at time ℓ_i . Since we use EDF principle, $d_{j'} \geq d_j \geq r_{u,opt}$ for all $j' \in \mathbf{Q}(\ell_i)$. There are two cases to set ℓ_i in \mathcal{S} : (1) the jobs in $\mathbf{Q}(\ell_i)$ will miss the deadline if the system is not turned on for processing at time ℓ_i , and (2) the energy-efficient anchor of a certain job is ℓ_i . For the first case, the sleep interval U_u cannot contain ℓ_i . Hence, when ℓ_i is contained in U_u , we have $d_j - \ell_i \leq \lambda\mathcal{B}$. Therefore, $r_i - r_{u,opt} \geq r_i - d_j \geq r_i - (\lambda\mathcal{B} + \ell_i) \geq (1 - \lambda)\mathcal{B}$. \square \square

Lemma 9. *Each sleep interval of \mathcal{O} intersects with at most two awaken-intervals of \mathcal{S} . In particular, each of U_1 and U_m intersects with at most one.*

Proof of Lemma 9. This lemma follows directly from Lemma 8. For $1 \leq i \leq \kappa - 2$, if r_i and ℓ_{i+1} are contained in a sleep interval $U_u = (\ell_{u,opt}, r_{u,opt})$ of \mathcal{O} , then $r_{i+1} - r_{u,opt} \geq (1 - \lambda)\mathcal{B} > 0$, which implies that ℓ_{i+2} will not be contained in U_u . \square \square

Lemma 10. For each awaken-interval I_i , $1 \leq i \leq \kappa$ and $\lambda \leq 1$,

$$\mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \leq \psi_\sigma \left(\sum_{j \in \mathbf{J}_i} c_j \right) + (3 - \lambda)E_w.$$

Furthermore, if $\mathcal{P}_i = 1$, then $\mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \leq 2E_w$.

Proof of Lemma 10. If $\mathcal{P}_i = 1$, then the total idle time in I_i is at most \mathcal{B} , since in our design, \mathcal{S} will turn off the processor immediately once $\mathcal{Q}(t)$ is empty when $t \geq \ell_i + \mathcal{B}$. Therefore $\mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \leq E_w + \psi_\sigma \cdot \mathcal{B} = 2E_w$, which is at most $\psi_\sigma \cdot \left(\sum_{j \in \mathbf{J}_i} c_j \right) + (3 - \lambda)\mathcal{B}$ when $\lambda \leq 1$.

If $\mathcal{P}_i = 2$, only processor M_2 could be in the standby mode according to our design. For the total standby time in I_i , we have two cases to consider: (1) If the system exits urgency after $\ell_i + \mathcal{B}$, then the total standby time on processor M_2 is upper-bounded by the workload on processor M_1 , which is at most $\sum_{j \in \mathbf{J}_i} c_j$. (2) If the system exits urgency before $\ell_i + \mathcal{B}$, the total standby time is at most \mathcal{B} , which is at most $\sum_{j \in \mathbf{J}_i} c_j + (1 - \lambda)\mathcal{B}$ since $\sum_{j \in \mathbf{J}_i} c_j \geq \lambda\mathcal{B}$ by Lemma 7. In both cases, the total standby time is at most $\sum_{j \in \mathbf{J}_i} c_j + (1 - \lambda)\mathcal{B}$. Therefore $\mathcal{E}_{w,\sigma}(\mathcal{S}, \ell_i, r_i) \leq 2E_w + \psi_\sigma \cdot \left(\sum_{j \in \mathbf{J}_i} c_j + (1 - \lambda)\mathcal{B} \right)$. By Lemma 7, we only have to consider $\mathcal{P}_i = 1$ and $\mathcal{P}_i = 2$, and this concludes the proof. \square \square

B.3 Jobs with Unit Execution Time

Lemma 13. When the jobs have unit execution times, for each awaken-interval I_i , $1 \leq i \leq k$, we have $\mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, \ell_i, r_i) \leq \left(3 - \frac{\lambda}{2}\right)E_w$.

Proof of Lemma 13. Since the total time in I_i is at most $(1 - \frac{\lambda}{2})\mathcal{B}$, we have

$$\mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, \ell_i, r_i) \leq 2E_w + \left(1 - \frac{\lambda}{2}\right)\mathcal{B} \cdot \psi_\sigma \leq \left(3 - \frac{\lambda}{2}\right)E_w.$$

\square

\square

Lemma 14. When $7 - \sqrt{41} \leq \lambda \leq 4 - \sqrt{10}$, we have

$$\mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, 0, \infty) \leq \left(3 - \frac{\lambda}{2}\right) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, 0, \infty) + \left(4 - \frac{\lambda}{2}\right) \cdot \mathcal{E}_w(\mathcal{O}, 0, \infty).$$

Proof of Lemma 14. We sketch only the main differences from the proof of Lemma 11 and adopt the remaining detail. For any awaken-interval I_i of \mathcal{S}^\dagger ,

- If I_i does not intersect with any sleep interval of \mathcal{S}^\dagger , then

$$\mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, \ell_i, r_i) \leq \left(3 - \frac{\lambda}{2}\right) \cdot E_w \leq \left(3 - \frac{\lambda}{2}\right) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, \ell_i, r_i).$$

- If I_i intersects with a sleep interval $U_u = (\ell_{u,opt}, r_{u,opt})$ and r_i is contained in U_u , we have two subcases. (a) If $\mathcal{P}_i = 1$, then

$$\mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, \ell_i, r_i) \leq 2E_w = 2\mathcal{E}_w(\mathcal{O}, \ell_{u,opt}, r_{u,opt}).$$

(b) If $\mathcal{P}_i = 2$, then $\mathcal{E}_{b,\sigma}(\mathcal{O}, r_{i-1}, \ell_{u,opt}) \geq \lambda E_w$ by Lemma 7. When $7 - \sqrt{41} \leq \lambda \leq 1$, we have

$$(1 - \lambda)E_w \leq \lambda \left(3 - \frac{\lambda}{2}\right) E_w.$$

Therefore,

$$\begin{aligned} \mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, \ell_i, r_i) &\leq \left(1 - \frac{\lambda}{2}\right) E_w + 2E_w \\ &\leq \left(3 - \frac{\lambda}{2}\right) \mathcal{E}_{b,\sigma}(\mathcal{O}, r_{i-1}, r_i) + 2\mathcal{E}_w(\mathcal{O}, \ell_{u,opt}, r_{u,opt}). \end{aligned}$$

- If I_i intersects with exactly one sleep interval $U_u = (\ell_{u,opt}, r_{u,opt})$ and ℓ_i is contained in U_u , then by Lemma 8, $\mathcal{P}_i = 1$ and $\mathcal{E}_{b,\sigma}(\mathcal{O}, r_{u,opt}, r_i) \geq (1 - \lambda)E_w$. When $0 \leq \lambda \leq 4 - \sqrt{10}$, we have

$$\frac{\lambda}{2} E_w \leq (1 - \lambda) \left(3 - \frac{\lambda}{2}\right) E_w.$$

Therefore,

$$\begin{aligned} \mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, \ell_i, r_i) &\leq 2E_w = \frac{\lambda}{2} E_w + \left(2 - \frac{\lambda}{2}\right) E_w \\ &\leq \left(3 - \frac{\lambda}{2}\right) \mathcal{E}_{b,\sigma}(\mathcal{O}, r_{u,opt}, r_i) + \left(2 - \frac{\lambda}{2}\right) \mathcal{E}_w(\mathcal{O}, \ell_{u,opt}, r_{u,opt}). \end{aligned}$$

From the above three cases, when $7 - \sqrt{41} \leq \lambda \leq 4 - \sqrt{10}$, we have

$$\mathcal{E}_{w,\sigma}(\mathcal{S}^\dagger, 0, \infty) \leq \left(3 - \frac{\lambda}{2}\right) \cdot \mathcal{E}_{b,\sigma}(\mathcal{O}, 0, \infty) + \left(4 - \frac{\lambda}{2}\right) \cdot \mathcal{E}_w(\mathcal{O}, 0, \infty).$$

□

□

B.4 Trading the Number of Processors with the Energy-Efficiency

Theorem 16. *Given a collection of k job sets, each satisfying Condition (1), we can compute a $\left(4 \cdot \max \left\{ \left\lceil \frac{k}{h-k} \right\rceil, 1 \right\} \right)$ -competitive schedule which uses at most h processors, where $h > k$, for the online power-minimizing scheduling problem.*

Proof of Theorem 16. The feasibility of the modified algorithm follows directly from our assumption on Condition (1) and Lemma 1.

For the competitive factor of the algorithm, it suffices to consider the case $k < h < 2k$. Consider the partition of the collection of job streams. Without loss of generality, let $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_{\lceil k/(h-k) \rceil}$ be one subset, and let $M_0, M_1, \dots, M_{\lceil k/(h-k) \rceil}$ be the processors allocated to these job streams. Consider any specific stream \mathcal{J}_i , $1 \leq i \leq \lceil k/(h-k) \rceil$, and the two processors M_0 and M_i . Observe that the lemmas presented in § 4.2 still holds for M_0 and M_i . Therefore the energy consumed by M_0 and M_i is bounded by four times the energy consumption of any offline optimal schedule for $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_{\lceil k/(h-k) \rceil}$. Since we have exactly $\lceil k/(h-k) \rceil$ such pairs, each unit of energy consumption in the offline optimal schedule, including switch-on operations, energy for executing jobs, and energy for standby, is charged at most $\lceil k/(h-k) \rceil$ times. Therefore, by taking all the pairs M_0 and M_i into account, we get the factor $4 \cdot \left\lceil \frac{k}{h-k} \right\rceil$ as claimed. □